

Documentation of MOCK-Package

Dennis Assenmacher, Christian Homberg January 4, 2016

1 Introduction

This documentation provides information about the R packages MOCK and PESA-II. Both packages were created as part of a seminar on advanced R in 2015/2016. The main goal of this documentation is to give insights in how the MOCK algorithm, proposed by Handl and Knowles [6] works in general, how it is implemented in R/C++ and how it can be used in order to get both analytical and visual results. Currently the packages are not available at the Comprehensive R Archive Network (CRAN). Therefore we suggest to install them via devtools:

devtools::install_github("Dennis1989/MOCK-PESA-II")

Within the following chapters we will explain the basic concepts of multi-objective clustering as well as the concepts of MOCK and different evolutionary multiobjective algorithms (EMOAs). We additionally provide some implementation details and how MOCK can be adjusted. Finally we focus on implementation details as well as benchmarking and conclude with working examples and provide an outlook.

2 Multi-objective clustering

Data clustering in general can be informally described as the process of finding homogeneous groups of data objects. The question which arises is how can we assess the quality of clustering solutions and therefore seperate good solutions from bad ones. One would argue that this is only possible by having access to external expert knowledge. Since clustering is an unsupervised learning method, such knowledge is rarely available. Therefore internal clustering criteria are applied. Handl and Knowles [4] describe it formally as an optimization problem:

$$P(C^*) = \min_{C \in \Omega} P(C)$$

where C is the set of all possible cluster solutions and $P: \Omega \to R$ an internal clustering criterion function.

In general one single criterion function would not be able to detect all possible cluster structures [4]. Therefore the need of reformulating the one-dimensional problem into a multi-dimensional optimization task is evident:

$$P_t(C^*) = \min_{C \in \Omega} P_t(C), t = 1, ..., n$$

The most prominent clustering critera are based on either achieving a compact solution or a connected one. Compact clusters try to identify solutions which hold the variation between cluster data as small as possible. In this context the k-means algorithm[8] and agglomerative clustering are frequently used. Connected clustering solutions on the other hand focus on grouping neighboring data together into one cluster. Average linkage tends to enforce the creation of those closely connected chains. However, it should be emphazised that it is not possible to detect all structures by only optimizing either compactness or connectedness. Therefore MOCK defines a multidimensional optimization task as stated above by optimizing in terms of compactness and connectedness. In the following section we will describe MOCKs evolutionary approach in detail.

3 MOCK

Basically MOCK can be separated into two distinct phases: The clustering phase and the Automatic determination of clustering solutions. In the following sections we describe each of those phases in detail.

3.1 Clustering Phase

First of all, the clustering phase uses an evolutionary algorithm in order to obtain distinct non-dominated cluster solutions which approximate the true pareto front. In case of MOCK, Handl suggests to use the pareto envelope-based selection algorithm version 2 (PESA-II). However, as a lot of different EMOA's currently exist, we try to describe the clustering phase more generally in order to be able to adapt MOCK to other EMOA's like SMS-EMOA or NSGA-II. A list of state of the art algorithms and their comparison can be inspected in [7].

All EMOAs follow the basic cycle of biological evolution: reproduction, mutation, recombination and selection. The difference between each algorithm lies in the operators used for these different steps. In order to be able to adapt MOCK to different EMOAs it is necessary to describe these operators as well as the problem representation and the objective functions used to determine the solution fitness.

3.1.1 Objective functions

As already described in the previous chapter, MOCK uses two contradicting clustering objectives based on compactness and connectedness.

To determine compactness, MOCK determines the overall deviation for each cluster solution. Informally it can be described as the sum of all summed distances between each data-object and their corresponding cluster center.

Formally we specify it as:

$$DEV(C) = \sum_{C_k \in C} \sum_{i \in C_k} \delta(i, \mu_k)$$

where C is denoted as the set of all clusters and μ_k the central point of one cluster C_k .

The second objective function, which measures the connectedness, is called connectivity. Informally it measures the degree to which neighboring data points have been placed in the same cluster [6]:

CONN(C) =
$$\sum_{i=1}^{N} \sum_{j=1}^{L} (x_{i,nn_{ij}})$$

$$x_{r,s} = \frac{1}{i}$$
, if $\notin C_k : r \in C_k$ and $s \in C_k$

where $nn_{i,j}$ is the j'th nearest neighbour of i. The formula indicates that the overall connectivity increases if neighboring points are not in the same cluster. Therefore the objective should be minimized as well as the overall deviation.

3.1.2 Representation

Each clustering solution is represented as a locus-based adjacency representation [6]. Each cluster solution contains N genes, where one gene is equivalent to one data-object within a certain cluster problem. Each gene can take a value $\in 1..N$. If gene i takes the value j, we can interpret it as an existing link between i and j. Since we can implement this within a simple numeric vector and also are able to simply apply mutation and crossover operators, this representation seems to work well for MOCK. Note that there is some computational effort necessary to determine which data objects belong to the same cluster. Handl argues that this decoding step can be achieved in linear time [6].

A visualisation of the locus based adjacency scheme can be found in 1.

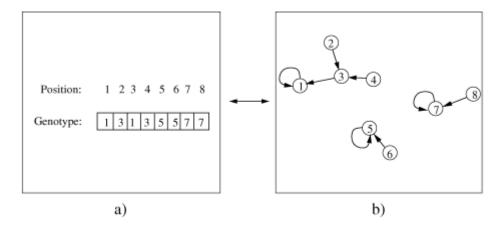


Figure 1: Locus-based adjacency scheme from [4]

3.1.3 Recombination

As recombination operator Handl uses uniform crossover. Two parents are first selected and a random masking vector is applied in order to determine which gene is transferred to the child.

3.1.4 Mutation

In order to reduce the search space for mutation, MOCK makes use of restricted nearest-neighbor mutation. The basic idea is to restrict possible mutations in a way that one gene can only be connected to one of its L nearest neighbors. A great advantage of this approach is that this drastically reduces the search space from N^N to L^N , where L < N and also saves a lot of computation time since the nearest neighbor matrix only has to be calculated once (we do not change the position of data objects). Additionally we have to consider that shorter links are more favourable than longer links. Therefore, Handl defines the probability of mutation for a link $i \to j$ as:

$$p_m = \frac{1}{N} + (\frac{l}{N})^2$$

where $j = nn_{il}$ (j is l'th nearest neighbor of i). It is obvious that a link $i \to h$ with nn would be preffered.

The probability of mutation can be directly integrated into the precomputed nearest neighbor matrix. More information on implementation details can found within chapter 5.

3.1.5 Initialization

An important step within the MOCK algroithm is the inital creation of initial solutions. The main goal is to achieve a well spreaded initial solution front in order to be able to search the whole solution space. As described in chapter 2, different single objective cluster algorithms perform well under either compactness or connectedness. Therefore, half of the initial solutions are generated with k-means, which performs well under compactness, and the other half by generating connected solutions obtained from a minimum spanning tree. In both cases the same amount of solutions is created by conducting k-means and respectively cutting the k-largest edge of the minimum spanning tree where k is iterated over 1 to maxCluster.

3.2 Automatic determination of clustering solutions

The previously introduced objective functions deviation and connectivity lead to a pareto front in two dimensional space. Since each point on the pareto front represents one non-dominated clustering solution (at least for the PESA-II implementation), it is not possible to identify one best solution in regard to the objective functions. Even more, the solutions differ in the amount of clusters (k), usually the deviation decreases as the connectivity and k increases. The solutions identified by MOCK during the clustering phase can therefore only be regarded as candidates which should be the object of further inspection, either by domain experts or by applying one of the many heuristic approaches for the determination of k in clustering theory. One approach which is also the basis for MOCK is to inspect the pareto front for a "knee", that is an area where the deviation greatly decreases while the connectivity does not increase much. Since the objective functions may be biased by the number of clusters, the inspection is conducted on a normalized pareto front. Furthermore, the MOCK pareto front is compared to reference fronts. These are generated by conducting MOCK on datasets with as similar characteristics as the original data as possible except for the existence of clusters. This data is generated uniformly within the eigenspace of the original data and back-transformed to its range. For each point of the pareto front the nearest Euclidean distance to any of the dominated control fronts is calculated, a measure called attainment score 9. It can be expected that a large attainment score indicates a good solution since deviation and connectivity are especially good when compared to solutions on the control fronts with the same amount of clusters. By the same logic a plot of attainment scores and amount of clusters can help identify a set of potential best solutions. The local maxima in this plot build a set of promising candidates while the global maximum can point to the best solution. [6]

4 Multi-Objective-Evolutionary-Algorithms (EMOA)

The following section provides a brief introduction into three different EMOAs, since these are included and can be used within the MOCK package. It should be emphasized that we will not provide a detailed description of each algorithm but only explain the fundamental concepts. As we already defined the operators used by MOCK, we can integrate them into each algorithm.

4.1 SMS-EMOA

SMS-EMOA was introduced in 2008 by Beume, Naujoks, and Emmerich [1]. The basic idea of this algorithm is to apply the S-metric within its selection operator. The S-metric is a measure which determines the dominated hypervolume of an approximated Pareto front. Therefore the S-metric should be maximized. This maximization is realized by eleminating the solution with the lowest contribution to the dominated hypervolume in each generation. In order to apply SMS-EMOA within MOCK, we have to integrate our predefined operators. First of all we have to make sure that we initialize as much solutions as we need. Since SMS-EMOA starts with a fixed set of solutions and only deletes and creates one new solution in each generation, the amount of initial solutions is equivalent to the amount of final solutions. Next we have to apply reproduction and mutation. This can be achieved by selecting randomly two parents from the last solution set and create a child by calling crossoverMock and mutating it via mutateMock, provided within the MOCK-package.

4.2 NSGA-II

In contrast to SMS-EMOAs $\mu+1$ selection strategy, NSGA-II [3] makes use of an $\mu+\mu$ selection, so that for each iteration μ new individuals are generated and μ individuals with the best fitness values are selected for the next generation. There are two different selection criteria within NSGA-II. First NSGA-II performs non-dominated sorting. Therefore all solutions are separated into different pareto layers. The second criteria is applied to the last non-dominated layer and is called crowded distance sorting. The crowding distance (CD) is a measure for the space around non-dominated solutions in which no other solution is found. All in all, applying CD enforces the selection of well spreaded solutions. For a detailed description of NSGA-II and the formalization of CD we suggest the original specification of NSGA-II [3].

4.3 PESA-II

In contrast to most of the standard EMOAs individual-based selection criteria, PESA-II follows a region-based approach that is the selection of a hyperbox rather than an individual [2]. Also two populations are stored during the run: an internal population (IP) of fixed size which is used for exploring new solutions and an external population of limited size which is used to maintain and exploit good solutions. Basically the algorithm follows distinct steps after creating the initial population and storing it into EP:

- 1. IP is filled by selecting randomly a populated niche from the hypergrid and a random solution s from that niche. Therefore highly populated areas do not contribute more than less populated areas.
- 2. All solutions in IP are processed by reproduction and variation.
- 3. If a solution in IP dominates any solution in EP, delete those dominated solutions from EP and add IP's solution to EP.

If a solution in IP is nondominated and EP is not full, add those solutions to EP. If EP is full only add the solution to EP if it is stored in a less crowded niche than any other solution. Therefore PESA-II encourages the exploration of the whole objective space. PESA-II is used within MOCK as the standard EMOA. One main advantage of PESA-II in that scenario is that IP determines the the amount of evaluations per generation and EP the maximum amount of final solutions. This separation is not present in the other algorithms.

Especially step three ensures the generation of a well spreaded pareto front since only those nondominated solutions are added to the external population which are situated within a less crowded niche.

4.4 Other EMOAs

As already indicated, it is straightforward to integrate other EMOAs into MOCK. The initialization of solutions, the representation and also the basic operators are already defined within the package and can be reused. All in all there are three steps which are necessary in order to adjust a EMOA.

- 1. Generate the necessary amount of inital solutions
- 2. Embed crossoverMock into the algorithm
- 3. Embed mutateMock into the algorithm

5 Implementation

In this chapter we explain some implementation details of the MOCK- as well as the PESA-II package. We start by giving an overview of the repository design followed by a discussion of how performance issues were addressed.

5.1 Repository overview

Currently we provide the possibility to run MOCK with either PESA-II, NSGA-II or SMS-EMOA. Since PESA-II is used within the original paper, it is defined as the stan- dard algorithm. As there was no R Package implementing PESA-II available when the seminar started, we decided to create an additional package PESA-II. Note that this package does not depend on the MOCK package and can therefore be used independently to solve any multiobjective optimization problem. Besides the two packages the github repository contains a third R folder additions that includes four files. These do not specifically belong to either package but are included in order to allow a reproduction of the results of this paper.

5.2 Performance improvements

Initially, both the MOCK objective and PESA-II were developed independently and entirely in R. Only the integration revealed that the performance of this implementation is insufficient. Using the profiler profvis (source: https://github.com/rstudio/profvis) we identified the most timeconsuming functions and implemented them in C++ using the Rcpp package. In the prioritization we also considered the estimated effort for the C++ implementation. Replacing one single line by dominatesC for example lead to speedups of several seconds. Below we list some of the most timeconsuming, frequently called, and important functions which were implemented in Rcpp:

- dominatesC
- fastNonDominatedSortingR
- decodeC
- deviationC
- connectivity
- whichC
- mutateC
- mockFunctionC

One of the most frequently called functions in all EMOAs is mockFunctionC which is why we chose to implement the whole objective function in C++. Still, since with the default parameters the function is called 10.000 times per MOCK run-through it takes about 20%-60% of the runtime, depending on the parameters chosen. Yet, only the partial implementation of fastNonDominatedSortingR in C++ came along with similarly large performance improvements for SMS-EMOA. The following function call took 94.2s using the improved function compared to 366.7s using the function included in the nsga2R package.

```
mock(data=clusterData,L=10,crossRae=0.7,nGrid=10,gens=10000,
method="smsemoa",controlFronts=1)
```

Even though the performance has already been drastically improved by implementing about 25% of the code in C++, it is clear that there is still room for improvements.

6 Benchmarking

Our benchmarking approach consists of three different methods:

- 1. Comparison of pareto fronts with original Mock tool (source to original tool).
- 2. Comparison of pareto fronts with other standard clustering algorithms.
- 3. Comparison of pareto fronts and clustering solutions between different EMOAs.

The methods and their results are presented in the subsequent subsections.

6.1 Comparison with original Mock tool

In order to legitimate our implementation of MOCK we compare pareto fronts obtained by the PESA-II implementation with the results of the original Mock tool by Handl. This visual benchmark is conducted for the same datasets and where possible for the same parameters. The results are depicted in figure 2.

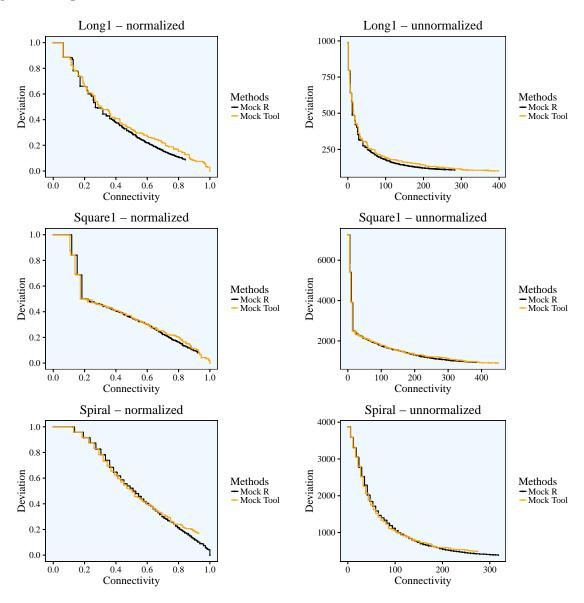


Figure 2: Comparison between MOCK package and original Mock tool

A superficial visual inspection of the benchmarks reveals that both implementations produce very similar pareto fronts, both for the unnormalized and the normalized plots. In the following we will only consider the unnormalized cases. A normalization without distorting the aspect ratio of each pareto front is not possible since usually the only common point in both fronts is the one cluster solution.

The dataset which produces the most similar results is Square1. In this case both pareto fronts overlap almost completely. On the first look, deviations between the fronts could be due to different

random numbers generated during runtime. However, this benchmark shows the same characteristics as the others, only much less distinct. For one, our R implementation mostly dominates the original Mock tool for solutions with many clusters while the Mock tool tendentially finds more and better few-cluster solution. Also, just as for the Long1 dataset, the Mock tool finds more solutions at the right lower end of the pareto front. However, these are partially dominated by the R solutions and this observation does not apply to the Spiral dataset. Besides different random numbers there are two more likely reasons why the implementations dont produce more similar pareto fronts. Firstly, some MOCK parameters used in the original tool are unknown and can therefore be only guessed. We default to the parameters proposed in [4] (with $k_{User} = 50$) which may differ from the actual implementation. Secondly, although our work is based on the pseudo-code and descriptions presented in [4] we cannot guarantee equality to the original implementation.

6.2 Comparison with standard clustering algorithms

Although we just showed that our MOCK implementation produces very similar solutions as the MOCK tool by Handl, as in the original paper[6], we compare our PESA-II implementation with the k-means algorithm as well as single and average linkage agglomerative clustering. Especially k-means (compactness) and single linkage agglomerative clustering (connectivity) perform well under MOCKs two objective functions and are therefore suitable for benchmarking the entire pareto front.

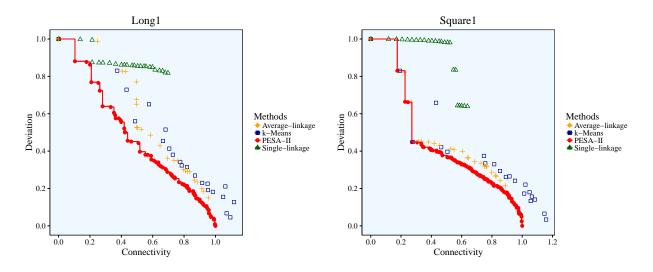


Figure 3: Comparison of pareto front with standard algorithms

The analysis of the results does not reveal new insights since for the Long1 and Spiral datasets the benchmarks resemble remarkably those presented by Handl and Knowles [5](see Figure 3). Given the very minor deviations to the original MOCK tool (see section 6.1) this was expected. Therefore we reference to the benchmarks applied previously by Handl and Knowles [6][5][4].

6.3 Comparison between different EMOAs

Originally MOCK was intended to be used in combination with PESA-II. Since we added two more algorithms, namely NSGA-II and SMS-EMOA, a performance evaluation would show which

algorithm works best. However, in order to achieve an objective result, we need to carefully specify MOCK's parameter for each individual algorithm. Because of their different selection strategies it is not possible to use the same number of generations. SMS-EMOA for example only generates one individual per iteration while NSGA-II generates μ new offsprings. Also we have to ensure, in terms of comparability, that the maximum number of solutions at our approximated-pareto front is the same for each algorithm. Figure 4 states the amount of evaluations per generation and the parameter which determines the maximum amount of solutions, for each of the three algorithms.

	PESA-II	NSGA-II	SMS-EMOA
#eval per generation	IPMax	μ	1
max solutions	EPMax	μ	μ

Figure 4: Comparison EMOAs: #evaluations and #solutions

By choosing one algorithm and setting fixed parameters we can adjust the settings for the other algorithms in a way that the overall amount of evaluations, namely the call of the mock function is the same for all of them. Therefore the recommended parameter setup for PESA-II within the original MOCK paper is used [4]. The following Figure 5 shows the parameter specification for each algorithm.

NSGA-II and SMS-EMOA do not differentiate between internal and external population as PESA-II does. Instead they specify μ in order to determine how many solutions will be available within the approximated pareto front. Using this specification we ensure that each algorithm executes 10.000 evaluations.

	PESA-II	NSGA-II	SMS-EMOA
#generations	1000	100	10000
EP	100	X	X
IP	10	X	X
gridsize	10	X	X
L	10	10	10
μ	X	100	100
p_m	0.7	0.7	0.7
#evaluations	IP * #gen	$\#gen * \mu$	#gen * μ

Figure 5: Parameter specification for each EMOA

Within the overall benchmark process we used three different datasets which were already analyzed in the latest publication of MOCK [4]: long1, square1 and spiral. Zitzler, Deb, and Thiele [9] propose three different quality-indicators for a multiobjective-optimization problem:

- 1. The distance of the approximated pareto front to the true pareto front should be minimized
- 2. Solutions should be well distibuted
- 3. For each objective a wide range of values should be achieved.

Since the true pareto front is unknown, we evaluate our solutions by comparing them to each other and to the solutions from the original MOCK-Tool. Figure 6 shows for each dataset the approximated pareto fronts of the three different algorithms in a normalized and unnormalized format.

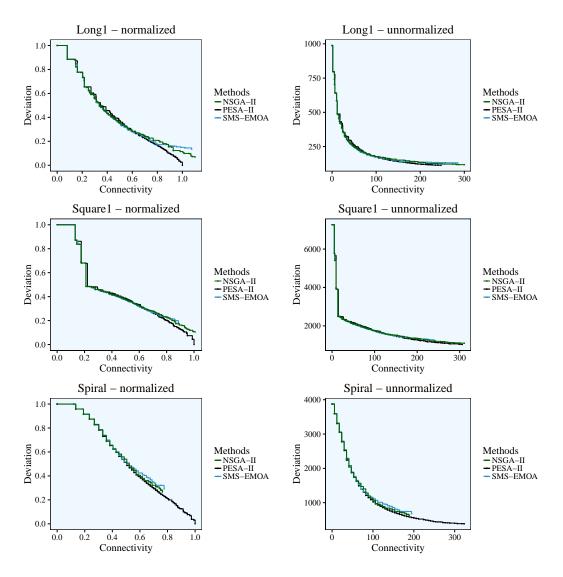


Figure 6: From top to bottom: long1, suqare1 and spiral dataset: normalized (left) and unnormalized(right) plots

At first sight none of those three algorithms' pareto fronts is completely dominated by any other. However at least for our test runs NSGA-II has the lowest share of the oveall pareto front for all three data sets. Also it is evident that PESA-II finds the best solutions for high number of clusters and outperforms the other ones in most cases.

By putting all 300 solutions into one single pareto front and deleting dominated points the overall contribution of each algorithm to this front can be determined. For the long and square dataset the contribution of SMS-EMOA algorithm to the overall pareto front is higher than the contribution of the other algorithms. However,

	PESA-II	NSGA-II	SMS-EMOA
long	37.64%	21.35%	41.01%
square	34.16%	16.77%	49.07%
spiral	47.54%	15.85%	36.61%

Figure 7: Share of solutions of overall pareto front

7 Usage

Within this chapter we want to provide guidelines of how to use the functions of the MOCK and PESA-II package by analyzing an exemplary dataset. For our purpose we use the Square1 dataset which was already analyzed in [4]. First of all we load the datset as a matrix into R:

```
square = read.csv("PATH\\square1.data",header = F, sep = " ")[,1:2]
```

Now it is time to run MOCK. As already described in the previous chapter, we first have to choose between one of the three different EMOA's which should be used during the clustering phase. First we decide to use PESA-II. Therefore we have to provide all necessary parameters. In our case we use the parameters used for analyzing the data in section six.

After this call we obtain a mock object which contains all information gathered through the run. Probably the most interesting attribute is the list of solutions. It can be accessed via calling

```
solution$sol
```

Each entry within the list represents one cluster solution and contains the corresponding encoding as well as its fitness values for deviation and connectivity. For post-processing purposes it is needed to decode the adjacency encoding to get the true cluster assignments for each data-object. Therefore we can decode the cluster by calling the decodeC function:

```
decodeC(solution$sol[[1]]$param)
```

However for a quick peak at the results (see Figure 8) we can also use the build-in visualization function of the MOCK package. To have a look at the approximated pareto front we can call

```
printParetoFront(solution,method="fronts",labelType="none")
```

Here we also include the reference fronts. If those additional fronts should not be displayed we can simply omit them by leaving out the method attribute. Next we can have a look at the attainment plot (9) in order to identify promoising cluster solutions. Therefore

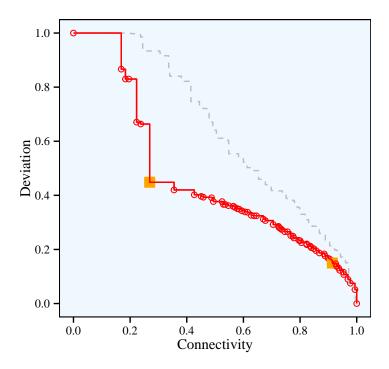


Figure 8: Approximated pareto front for Square1 dataset

```
printAttainmentScore(solution)
```

has to be called. We can directly see the most promoising solutions visually indicated by their red color with their corresponding solution index. If we now want to look at the best solution (see Figure 10), we can simply plot solution number seven by calling:

```
printClusterSolution(solution,7,square,1)
```

Now we can also run the same problem for a different algorithm like SMS-EMOA. Keep in mind that the parameters have to be adjusted accordingly. We again use the parameter setting of section 6:

Now we have two solution objects. If we want to compare their pareto fronts (see Figure 11) we can use another function within the MOCK package:

```
printMultipleParetoFronts(list(solution,smsSolution))
```

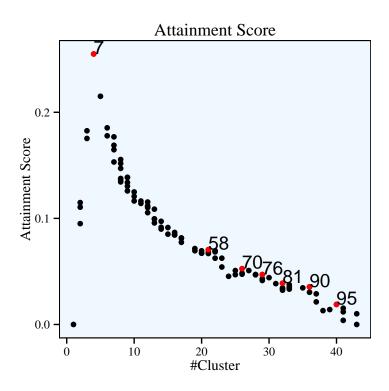


Figure 9: Attainment score with the global and local maxima

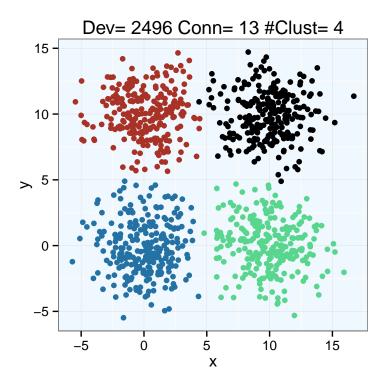


Figure 10: Visualization of the best solution for Square1 dataset

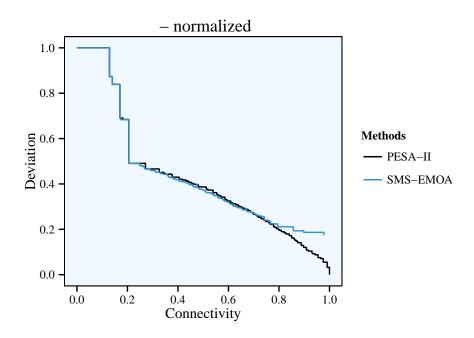


Figure 11: SMS-EMOA and PESA-II pareto fronts for the Square1 dataset

8 Outlook

All in all the MOCK and PESA-II packages provide tools to analyze data structures efficiently. Next to PESA-II two additional EMOAs can be used for the clustering phase. Also it is possible to adapt other EMOAs by using the recombination and mutation operators. A plethora of vizualization methods allows the analyst to get a fast understanding of the

However in terms of the algorithm's runtime there is still space for improvements as it was already indicated within the implementation section. Although most of the crucial high volume functions were implemented in C++, especially the creation of many reference fronts is still pretty timeconsuming. Since the mock run on the clusterdata as well as the mock runs on the reference data is totally independend on each other, those runs can be parallelized. For upcoming versions of the package this feature is planned to be implemented with the highest priority. In regard to performance, no matter which EMOA is used for MOCK, the R implementation cannot compete with the speed of the original Mock tool. This is because the latter is implemented in C, a language much closer to machine language. After all the goal of the seminar was to provide an R implementation of MOCK, which currently does not exist, so that all the advantages could be gained. One of which is the integration of the package into R which allows easy pre- and post-processing and eliminates the effort to create an interface between the original Mock tool and other statistical computation tools.

References

- [1] Nicola Beume, Boris Naujoks, and Michael Emmerich. "SMS-EMOA: Multiobjective selection based on dominated hypervolume". In: *European Journal of Operational Research* 181.3 (2007), pp. 1653–1669.
- [2] David W Corne et al. "PESA-II: Region-based selection in evolutionary multiobjective optimization". In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2001*. Citeseer, 2001.
- [3] K. Deb et al. "A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II". In: *Parallel Problem Solving from Nature PPSN VI*. Ed. by Marc Schoenauer et al. Berlin: Springer, 2000, pp. 849–858.
- [4] Julia Handl and Joshua Knowles. "An evolutionary approach to multiobjective clustering". In: Evolutionary Computation, IEEE Transactions on 11.1 (2007), pp. 56–76.
- [5] Julia Handl and Joshua Knowles. "Exploiting the trade-offthe benefits of multiple objectives in data clustering". In: *Evolutionary Multi-Criterion Optimization*. Springer. 2005, pp. 547–560.
- [6] Julia Handl and Joshua Knowles. "Multiobjective clustering with automatic determination of the number of clusters". In: *UMIST*, *Manchester*, *Tech. Rep. TR-COMPSYSBIO-2004-02* (2004).
- [7] Daniel Kunkle. "A summary and comparison of MOEA algorithms". In: (2005).
- [8] James MacQueen et al. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297.
- [9] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. "Comparison of multiobjective evolutionary algorithms: Empirical results". In: *Evolutionary computation* 8.2 (2000), pp. 173–195.